# Course reader: *Plotting*

- The importance of data visualization cannot be understated, particularly in signal processing and data analysis. Visualizing data will help you learn equations and algorithms, understand signal processing steps, and help you find patterns in data, as well as errors, outliers, and other non-representative data points. MATLAB has a nice graphics engine and you should use it as much as possible!

- There are many many possibilities for visualizing 2D and 3D data (and even 4D data when you include movies!). Once you learn the basics, using more advanced plotting isn't so difficult.

- Graphics go into figures, and parts of figures called subplots. You can create a new figure by typing `figure` or `figure(3)` to create figure #3. Plots are drawn into the "active" figure, which is the one most recently mouse-clicked or called from the command.

- The `plot` function is for making lines or dots. The basic argument is `plot(y)`. You can also embellish your plot by changing line color, marker shapes, and so on. For example:

```
plot(1:8,exp(1:8),'ks-','linewidth',3,...
   'markerfacecolor','w','markeredgecolor','m','markersize',20)
```

- Each time you call the `plot` or any other plotting function, the previous plot will disappear. You can type `hold on` to plot on top of a previous plot. Watch:

```
figure
plot(randn(20,2))
plot(randn(20,1)) % previous lines gone!
hold on
plot(randn(20,1)) % old line stays
hold off
plot(randn(20,1)) % old lines are gone again
```

- `bar` makes a bar plot, and `errorbar` can be used to put error (variance) lines, like this:

```
figure
m = [1 3 2 5 -3];
e = [1 1 .1 3 .3];
bar(1:length(m),m) % first input is optional
hold on
errorbar(m,e,'.')
```

- `imagesc` and `contourf` are two commonly used functions for viewing 2D data. `contourf` is smoother, and the y-axis is reversed in `imagesc`.

```
pic = imread('moon.tif'); % comes with MATLAB
figure
subplot(121) % subplot geometry: 1 row, 2 cols, plot #1
imagesc(pic), title('using imagesc')
```

```
subplot(122)
contourf(pic,100,'linecolor','none'), title('using contourf')
```

- MATLAB native figure format is .fig. You can save figures using the *Figure* menu or with code: `savefig('figurename.fig')`. To save a figure as an image file, use `print`:

  ```
  figure
  plot(randn(50,3))
  print('filename.png','-dpng') % creates png
  ```

- The `get` function will get properties of a figure, axis, or plot. A typical call is `get(gca)` where `gca` means "get current axis."

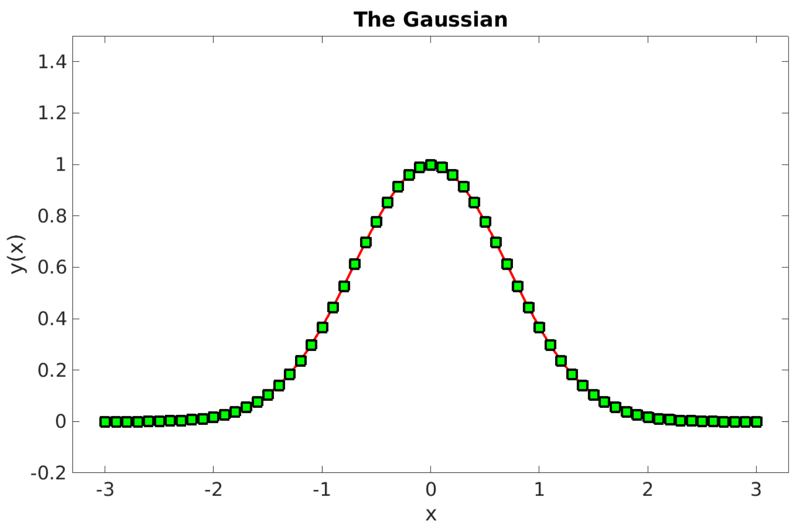
- The `set` function allows you to change features of figures, axes, and plots. For example:

  ```
  figure
  plot(randn(20,2))
  set(gca,'xlim',[-3 49]) % change the x-axis limits
  set(gcf,'color','m','name','my figure')
  ```

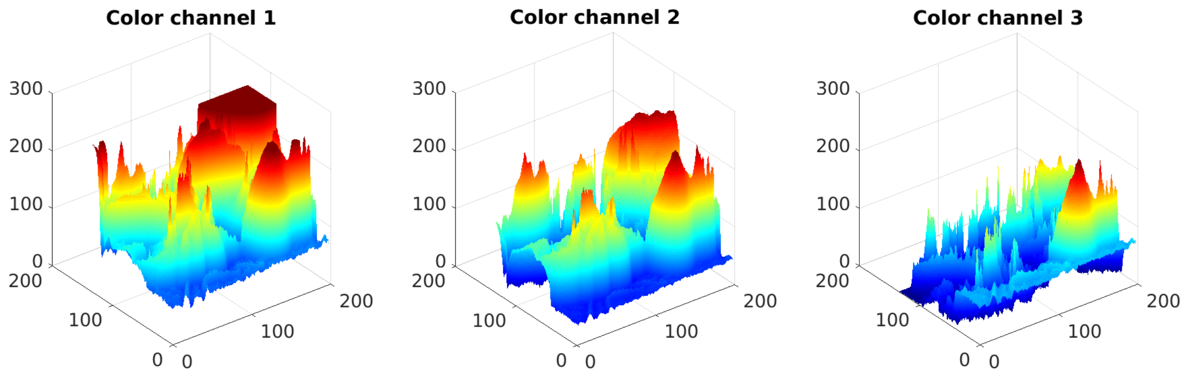
- You can also make a lot of plot adjustments via the MATLAB figure window. But I don't recommend this—a key advantage of programming is *reproducability*. Manual adjustments are difficult to reproduce, but if all features are done using the `set` and other commands, then you (and anyone else) will be able to recreate the plot in the future.

## Exercises

1. The figure below shows the function $y = e^{-x^2}$ (the MATLAB function for the natural exponent $e^x$ is `exp(x)`.) Recreate this figure as best you can.



2. Same goal, different picture. This one uses the 'onion.png' picture that is built-in with MATLAB. *Hints*: look up `surf` and `shading`, and check the size of the dimensions of the image.



3. To draw a straight line in MATLAB, specify the start and end points, like this: `plot([0 1],[-4 5],'k')` Use this knowledge to create a square on the top subplot and a pentagon on the bottom subplot (doesn't have to be perfectly geometrically precise).
4. You can embed `get` inside `plot`. This is often used to create a line across the entire axis. Create a plot of 100 normally distributed random numbers, and then use `set` to plot a thick black line at y=0 across the entire x-axis.

## Answers

**1.** Note that the line width and marker size numbers depend on monitor resolution, so don't worry if you have to use different values for those parameters.

```
x = -3:.1:3;
plot(x,exp(-x.^2),'rs-','linewidth',3,...
   'markerfacecolor','g','markersize',15,'markeredgecolor','k')
xlabel('x'), ylabel('y(x)')
set(gcf,'color','w')
set(gca,'ylim',[-.2 1.5],'xlim',[-3.3 3.3])
title('The Gaussian')
```

**2.** .

```
pic = imread('onion.png');
for i=1:3
   subplot(1,3,i)
   surf(pic(:,:,i))
   shading interp
   axis square
   title([ 'Color channel ' num2str(i) ])
end
```

**3.** Here's a solution.

```
subplot(211)
plot([0 1],[0 0],'k'), hold on
plot([1 1],[0 1],'k')
plot([0 1],[1 1],'k')
plot([0 0],[0 1],'k')
axis([-1 2 -1 2]), axis square

subplot(212)
plot([0 1],[0 0],'k'), hold on
plot([1 1.5],[0 1],'k')
plot([1.5 1],[1 2],'k')
plot([0 1],[2 2],'k')
plot([-.5 0],[1 2],'k')
plot([-.5 0],[1 0],'k')
axis([-1 2 -.5 2.5]), axis square
```

**4.** This problem highlights an important tool in programming: embedding one function in another function, thus making the *output* of one function be the *input* to another function.

```
plot(randn(100,1)), hold on
plot(get(gca,'xlim'),[0 0],'k','linew',3)
```